Brandon Wong
CS506 Midterm

Predicting Star Ratings from Amazon Movie Reviews

Data Preprocessing and Exploratory Data Analysis (EDA)

During my exploratory data analysis, I noticed several key patterns:

1. **Skewed Distributions**: I made the helpfulness feature in order to reduce the dimensionality of the HelpfulnessNumerator and HelpfulnessDenominator features. But I noticed when I graphed its distribution, the Helpfulness feature was highly skewed, with values ranging widely.
2. **Imbalanced Target Variable**: I visualized the distribution of Score to understand its distribution (*Figure 1*). I also examined correlations between numerical features and the target variable to identify potential predictors. From the graph I noticed that Score, was not evenly distributed; certain star ratings were more prevalent, with the majority of the data being 5 or 4 stars.
3. **Duplicate Entries**: Multiple reviews shared the same UserId and ProductId, indicating potential duplicates or multiple reviews by the same user for the same product. So I thought because of this I could use that duplication to build a history around the User and Product to see their patterns and reviews.
4. **Missing Data Handling**: I calculated the percentage of missing values for each column. The missing data mostly resided in the variables Text and Summary , but the majority of columns weren't missing any information.
5. **Sentiment Segmentation**: Preliminary text analysis revealed that high-scoring reviews often contained different vocabulary and sentiment compared to low-scoring ones (*Figure 2*). This insight highlighted the potential value of sentiment as a feature or using popular words as flags for positive/negative assumptions. But I also did notice that some popular words did overlap in both high and low rated reviews

Feature Engineering and Transformation

To improve the model's performance and capture meaningful patterns, I engineered new features and transformed existing ones.

1. **Helpfulness Log Transformation**
   As mentioned earlier I made the helpfulness feature, but the Helpfulness feature's skewness could skew the model (*Figure 3*). So I applied a logarithmic transformation (log1p) to normalize its distribution. This reduced the impact of outliers and made the feature more suitable for modeling.
2. **Sentiment Analysis on Review Text**
   Recognizing that sentiment could be a strong predictor of star ratings, I performed sentiment analysis on the Summary and Text fields using the VADER tool which I read about in a Medium article. By extracting compound sentiment scores, I captured the overall sentiment polarity of each review. This helped distinguish between positive and negative reviews, enhancing the model's predictive power.
3. **Encoding User and Product IDs for History**
   I encoded categorical variables like *UserId* and *ProductId* using Label Encoding. I ensured that the encoding was repeatable by *pickling* the encoding and implemented a safe transformation by assigning a special index to any new IDs encountered in the test set. This ensured consistent

encoding across training and testing phases. I used this to identify the same product or user for training. By using a Groupby I can train on their history.

4. **Aggregation of Product-Level and User-Level Features**
   To capture high level patterns at the product and user levels, I created aggregate features.These aggregates allowed the model to consider habitual behaviors and inherent product qualities, improving its ability to generalize. I will later apply these averages to my test sets by joining on products and users.
   a. Product-Level Aggregates: For each product, I calculated the average rating, total number of reviews, average helpfulness score, and average sentiment scores.
   b. User-Level Aggregates: For each user, I computed their average rating given, the number of reviews written, and their average sentiment scores.

Model Selection and Hyperparameter Tuning

1. **Choice of LightGBM and External Resources**

I selected LightGBM (LGBMClassifier) for its efficiency with large datasets and gradient boosting capabilities, which help prevent overfitting and improve generalization. It is also optimized for speed so it would make training a lot quicker than say XGBoost. To optimize the model effectively, I referred to the article "Complete Guide on How to Use LightGBM in Python" by Analytics Vidhya. This resource provided valuable insights into LightGBM's parameters and advanced features. I learned an advantage was that it still worked effectively even if variables weren't scaled and that was great for dealing with large number ranges. From the article, I structured my code to incorporate best practices and optimized the model more effectively. My model utilized the features:

```
['Score', 'Year', 'Helpfulness_log_scaled', 'Summary_sentiment',
'Text_sentiment', 'avg_user_score', 'num_user_reviews', 'avg_rating',
'num_reviews', 'avg_helpfulness', 'avg_summary_sentiment',
'avg_text_sentiment', 'ProductId_encoded', 'UserId_encoded']
```

2. **Train, Validation, and Test Split**

To prevent overfitting and ensure robust hyperparameter tuning, I split the dataset into training, validation, and test sets using an 70/15/15 split. I used the training set to train my model on. Then to test different hyperparameters, I utilized the validation to verify I was tuning my model in the right direction. Then finally when I believe I achieved good results, I would test it on the test split and this helped avoid overfitting to the training and validation data. This also gave me the opportunity to analyze misclassifications to understand where the model struggled, identifying potential areas for improvement.

3. **Hyperparameter Tuning**

Using RandomizedSearchCV with five-fold cross-validation on the combined training and validation sets, I optimized the following hyperparameters:

- **n_estimators**: Number of boosting rounds (tested values from 100 to 500).
- **learning_rate**: Shrinks the contribution of each tree (explored values between 0.01 and 0.2).
- **max_depth**: Maximum depth of a tree (adjusted between -1 and 15).
- subsample and colsample_bytree: Fraction of observations and features to use (tuned to improve generalization).

- **num_leaves**: Maximum number of leaves in one tree (modified to control tree complexity).

This resulted in the Best Parameters: {'subsample': 0.9, 'num_leaves': 70, 'n_estimators': 500, 'max_depth': 5, 'learning_rate': 0.1, 'colsample_bytree': 0.8}. My test score after tuning was 0.642 and my testing score was 0.643 meaning that it was well fitted with minimal loss between my validation and test

4. Key Assumptions

a. Validity of Helpfulness Scores: I assumed that the Helpfulness scores provided by users are reliable indicators of the quality and usefulness of reviews.

b. Sentiment Reflects Satisfaction: I presumed that the majority of sentiment scores derived from review text correlate with the user's satisfaction and, consequently, their star rating. However there are rare times where they dont always align or are too neutral to pick a direction.

c. Consistency in User and Product Behavior: I believed that users and products exhibit consistent patterns that can be captured through aggregated features.

References

1. Analytics Vidhya: "Complete Guide on How to Use LightGBM in Python"
   Source:
   https://www.analyticsvidhya.com/blog/2021/08/complete-guide-on-how-to-use-lightgbm-in-python/
2. Rslavanyageetha: "VADER: A Comprehensive Guide to Sentiment Analysis in Python"Source:
   https://medium.com/@rslavanyageetha/vader-a-comprehensive-guide-to-sentiment-analysis-in-python-c4f1868b0d2e

FIGURES

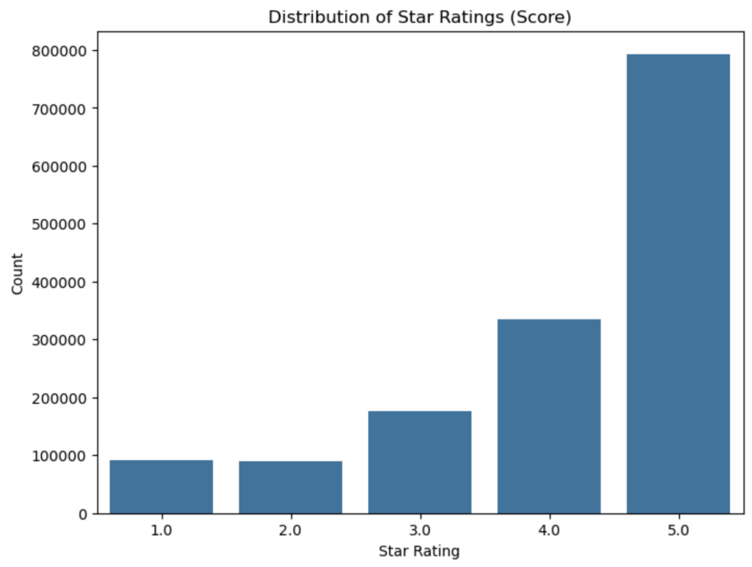


Figure 1.

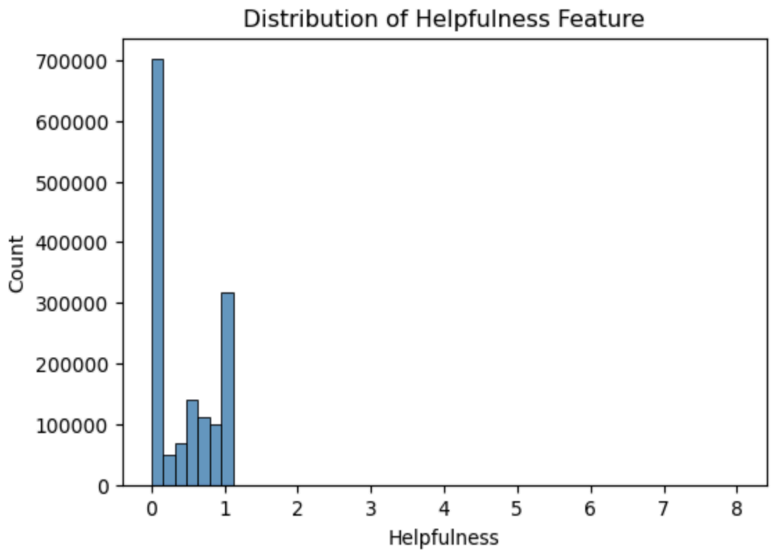
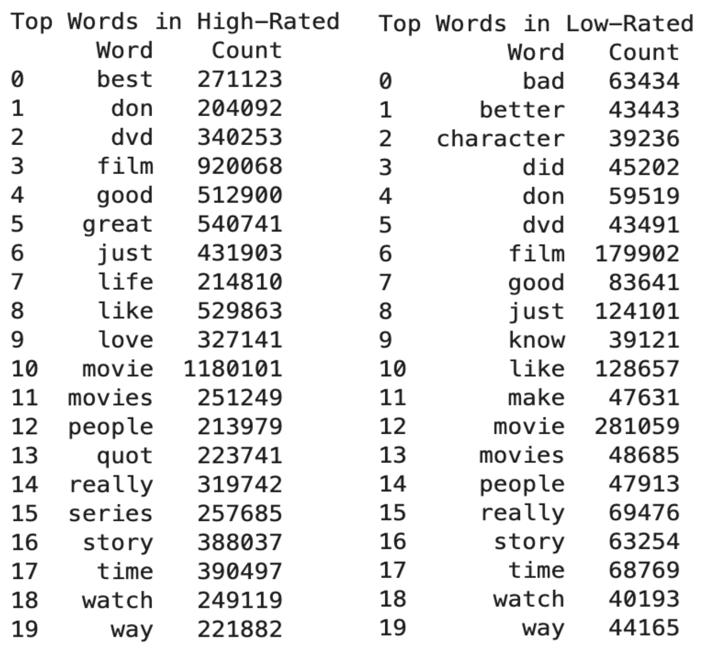


Figure 3.



```
Top Words in High–Rated
      Word    Count
0     best   271123
1      don   204092
2      dvd   340253
3     film   920068
4     good   512900
5    great   540741
6     just   431903
7     life   214810
8     like   529863
9     love   327141
10   movie  1180101
11  movies   251249
12  people   213979
13    quot   223741
14  really   319742
15  series   257685
16   story   388037
17    time   390497
18   watch   249119
19     way   221882
```

```
Top Words in Low–Rated
         Word   Count
0         bad   63434
1      better   43443
2   character   39236
3         did   45202
4         don   59519
5         dvd   43491
6        film  179902
7        good   83641
8        just  124101
9        know   39121
10       like  128657
11       make   47631
12      movie  281059
13     movies   48685
14     people   47913
15     really   69476
16      story   63254
17       time   68769
18      watch   40193
19        way   44165
```

Figure 2.